# Project Description

In this project, we sort a list of 10,000 names by alphabetical order using parallel sorting technique and sequential sorting. The main focus is to observe the efficiency improvement by using parallel sorting teqnique in terms of sorting implementation time.

In order to compare the letters in names, we first modified the built-in class function CompareTo() which is orginally only apply to numerical values. Later on, We will use the modified CompareTo() function to do the comparison between different letters in names after parallel sorting for swap purpose.

In the ParallelSort class, we mainly implement a quick sort algorithm. We defined a Partition function to prepare the sorting list for the recursive sorting calls. This includeds the CompareTo() function to do the swap and move the start index of child list to the left. Swap function is also defined seperatly in this class.

The QuicksortParallel function is the sorting algorithm part. In this function, we first defined the condition to use either QuicksortSequential or QuicksortParallel functions. The key part is in the case of QuicksortParallel. We first call the Partition function to get the pivot element index in the parent list. Next, we create two threads for the recursive calls of the QuicksortParallel on the left side list and right side list using the pivot value. This is the parallel sorting process. The whole recursive calls will begin to return having the partitioned list once the child lists only contian one element.

In the main program class, we first read in the name list and prepare it for sorting. Then we start the timing before we call the QuicksortParallel function and stop the timing and print the time after the sorting call.

The result in our case is not very ideal. We actually record a longer time for parallel sorting than the sequential sorting. We believe this is because of the overhead time in the scheduler and also based on the fairly small task we have in this case.